Massively Parallel Distributed Computing: World's First 281 Gigaflop Supercomputer

Jerry Bolen, Arlin Davis, Bill Dazey, Satya Gupta, Greg Henry, David Robboy, Guy Schiffler, David Scott, Mack Stallcup, Amir Taraghi, Stephen Wheat
- Intel Corp.
LeeAnn Fisk, Gabi Istrail, Chu Jong, Rolf Riesen, Lance Shuler
- Sandia National Laboratories

Abstract

This paper describes the Intel and Sandia joint effort to combine two Intel MP Paragon™ supercomputers via multiple HiPPI channels. This effort broke the world speed record on the massively parallel LINPACK benchmark by approximately fifty percent, attaining 281.1 GigaFLOPS. The two MP Paragon supercomputers operated under the SUNMOS operating system.

Keywords: HiPPI, World record breaking, SUNMOS, LINPACK benchmark

1 Introduction

We describe the hardware, the operating system, the MPLINPACK application, and the HiPPI subsystem utilized in the Intel and Sandia joint effort to combine two Intel MP Paragon™ supercomputers via multiple HiPPI channels. We then focus on the environment used to allow the two Intel supercomputers, connected via HiPPI, to operate as a single supercomputer and describe the optimizations made in MPLINPACK to achieve this goal. Finally, we show some modeling of the predicted performance and compare this to our actual results.

1.1 Hardware

The Intel MP Paragon™ XP/S supercomputer consists of a two-dimensional mesh of nodes arranged into cabinets of size 16 high by 4 wide [11]. Each node contains three Intel i860® XP RISC microprocessors, capable of 75 MFLOPS [1] double precision each. Nodes communicate across the interconnection network at 175 Mbytes/second. Latencies are typically in the range of 30 microseconds, with special kernels capable of running less than 10 microseconds [13].

Two Intel MP Paragon supercomputers were used in this effort. One, the machine purchased by Oak Ridge National Laboratories, had 1024 MP nodes. The other had 1232 MP nodes. The combined configuration brought on-line 2256 MP3 nodes, or 6768 i860® processors. Each MP Paragon supercomputer had 16 HiPPI nodes connected via HiPPI channels to the other MP Paragon system.

1.2 Operating System

The two systems were running the SUNMOS S1.6.2 operating system. SUNMOS (Sandia and University of New Mexico Operating System) is a joint collaboration between Sandia National Laboratories and the University of New Mexico. SUNMOS is copyrighted by Sandia National Laboratories. [2] SUNMOS is a high performance operating system with low memory overhead and delivers a high message bandwidth. SUNMOS allows application codes to use all three processors on each node for computation, as compared to the current release of Paragon/OS, which uses one of the three processors as a message co-processor.

1.3 The Code

The demonstration was performed both with the MP LINPACK benchmark, which is a well known measure of comparison between high performance supercomputers, and a complex double precision variant of the code. Although the complex version of the code

[1] MFLOPS refers to millions of floating point operations per second. In single precision, the Intel i860® XP is capable of 100 MFLOPS.

[2] SUNMOS questions can be forwarded to sunmos-support@cs.unm.edu.

(henceforth: ZLU) is not a controlled benchmark like MPLINPACK, ZLU is arguably as significant, since a large number of real world problems require the solution of a large linear system of complex equations. The LINPACK benchmark measures the time it takes to solve a real (double precision, 64 bits) linear system of equations of size N with a single right-hand side [3] [2].

Performance is measured in terms of billions of floating point operations per second (GFLOPS), which is $2N^3/3 + 2N^2$ where N is the number of unknowns (or $8N^3/3$ for ZLU.) FLOPS reported are real FLOPS and not "macho" FLOPS based on Strassen [14] multiplication. Explicit residual bounds were computed to verify accuracy.

1.4 The HiPPI Subsystem

Communication between the two systems is via bidirectional HiPPI connections directly from one system to the other. Each system has 16 HiPPI nodes, connected pairwise with the HiPPI nodes in the other system using a cable in each direction, a total of 32 cables. The 16 bidirectional HiPPI connections correspond logically to one connection per row of compute nodes; so, in a sense, the HiPPI connections extend the mesh horizontally across two machines. Each HiPPI node runs the SUNMOS operating system, which includes a HiPPI driver. A HiPPI server runs as a SUNMOS application on each HiPPI node.

On the sending side, sending messages across the HiPPI line uses a typical I/O paradigm which includes commands to open the device, send messages, and close the device. To send a message over the line, the data is prepended with appropriate header information and sent via the mesh to the server on a HiPPI node on the same machine (See Figure 1). The header information includes the physical node address of the final destination compute node.

On the receiving side, the HiPPI server does an automatic store-and-forward to the destination node. Thus the destination compute node does not use an I/O paradigm, rather it receives a SUNMOS message on the mesh, which arrives from the HiPPI server instead of from a compute node. The store-and-forward mechanism is optimized so that the message is sent to the mesh directly from the HiPPI buffer, it is not copied to user space on the HiPPI node before being forwarded. This model is demonstrated in Figure 1.

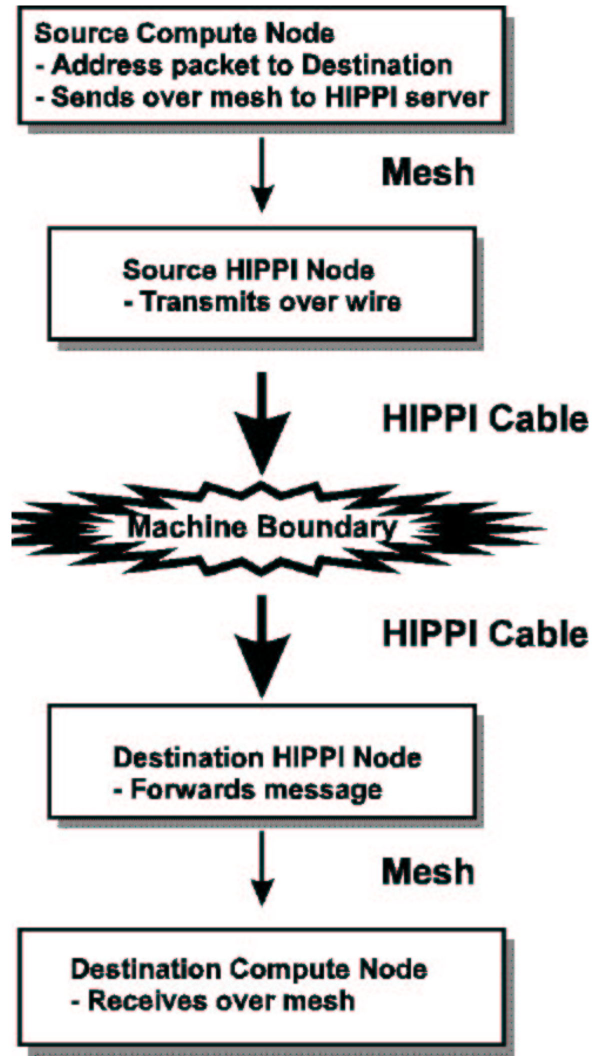

Figure 1: HiPPI Subsystem

To measure the performance of the HiPPI subsystem, we sent a message from machine 1 to machine 2 and back again, measuring the overall round-trip time on one system. We observed 22 Mbytes/sec overall on a one megabyte message, which implies 44 Mbytes/sec one way. There is about 1 milliseconds round trip latency.

More recent experiments were done by Lance Shuler, who is optimizing the HiPPI driver at Sandia. He had a one-way experiment that yielded bandwidths exceeding 50 Mbytes/sec for messages of 128 Kbytes, using a single HiPPI node. When several messages were sent at the same time, overall performance improved.

[3] The latest copy of the MPLINPACK benchmark report can be obtained via anonymous ftp to netlib.att.com in directory netlib/benchmark. The relevant file is in binary compressed mode and is called performance.ps.Z. Equivalently, one could send e-mail to netlib@att.com and request "send performance from benchmark".

2 The Dual Machine Environment

The front end for the MPLINPACK code was changed so that there are two separate executables, for the "left" and "right" machines. Configuration files tell each machine the layout of HiPPI nodes and compute nodes for both machines.

At the communication level, when a node needs to send data to another node, it calls a communication function with the address of the destination node. This function was modified to decide whether the destination is local or remote. In case it is remote, the routine looks up the destination physical node address in tables, and invokes code to transmit the message. This routine encases the message in headers, looks up the address of the HiPPI node corresponding to the sender's row in the mesh, and sends the message to the HiPPI server on the HiPPI node, which sends it over the wire. In order to prepend the header, short messages are copied to a buffer that includes the header. Long messages have the header prepended to them in place, then the contents of the memory preceding the message is restored.

On the receiving side, the HiPPI server gets the destination node address out of the packet and forwards the message. The receiving compute node does not do anything special, it receives a SUNMOS message from the HiPPI node as it would from another compute node. Tables are used to configure the locations of the HiPPI nodes and the compute nodes on both systems. All of the HiPPI nodes on each side can operate in parallel independently of each other.

To summarize the message path:

- Local messages travel over the mesh with essentially no change in the logic.
- For remote messages, the path is

$$\text{source} \stackrel{\text{mesh}}{\Longrightarrow} \text{HiPPI server} \stackrel{\text{wire}}{\Longrightarrow}$$
$$\text{HiPPI server} \stackrel{\text{mesh}}{\Longrightarrow} \text{destination}$$

Remote messages travel on the interconnect of each machine in addition to the time required over the HiPPI wire. For example, suppose the message on machine 1 travels at 150 MB/sec to HiPPI 1, and the message on machine 2 goes from HiPPI 2 to the compute node at 150 MB/sec. If the HiPPI wire represented "instantaneous" communication, then the overall performance would be bounded by 75 MB/sec. This communication scheme is scalable. That is, nodes can independently and concurrently gain similar bandwidths for high aggregate rates.

2.1 MPLINPACK Optimizations

MPLINPACK uses a block two dimensional wrap mapped data decomposition [7]. With a two dimensional data decomposition, messages flow naturally in two directions: horizontal and vertical. Horizontal messages involve the bulk of the communication. As such, they are pipelined in a ring and overlapped with the computation. We refer to a node logically by zero-based row and column number and denote it as such: (row,column). If there are 10 columns in the left machine, we denote the top left node of the right machine as (0,10).

Suppose column 8 of the left machine ((*,8)) is originating a message to be broadcast horizontally. For purpose of illustration, we assume that the logical partitioning of nodes in the algorithm corresponds to the actual physical locations, although in general that does not need to be the case. By virtue of the data decomposition, each row has a separate portion of the message. For example, nodes in row 0 need to have access to the data that (0,8) originated, but not (1,8). Therefore, if there are 16 rows (which is the case when the logical mapping matches the physical mapping), then there are really 16 separate messages all moving horizontally to the right at the same time in a ring fashion. That is, node (0,8) sends the message to (0,9), which in turn sends the message to (0,10), etc..

The horizontal broadcasts move rightward from a source column and continue in a ring until all the columns have the message. The source column changes during the course of the algorithm and may be on the left machine or may be on the right machine. When the broadcast reaches the right end of the left machine, the message is sent over the mesh to the 16 HiPPI nodes (which could be anywhere in the machine) and then sent over the HiPPI wires to the HiPPI nodes on the right machine. Finally, the message is forwarded to the left end of the right machine. The same applies when the broadcast reaches the right end of the right machine (it is then sent via mesh to the HiPPIs then through the mesh to the left end of the left machine.) The first observation is that only nodes along the left and right columns of the both machines ever need to use the HiPPI connection. The model we presented was more general than that because we achieved the additional generality for free.

We found it crucial not to send too many messages over the HiPPI wires at once since the HiPPI server we were using had a fixed amount of buffer space. Handshaking partially solved this problem. A node would post an irecv for the incoming data, then send a message saying that it is ready to receive the data. The sending node would not send the message until it received the message saying it was okay to send it. In the mean time, the receiving node might be doing other work. When the receiving node needed to get the new data, it would block until the data was in place. This handshaking would prevent data from subsequent messages from piling up until the receiving node is ready to use them. Since the sending nodes usually get the send ready before the nodes to the immediate right are ready, the first set of handshaking is avoided.

Another observation is in the course of moving rightward, eventually a horizontal broadcast originating in the center of the left machine must eventually pass through the right machine and then back to the left part of the left machine (or vise versa.) A more efficient broadcast from a source machine would send the data to the other machine once, and let each machine do its own local broadcast. Instead, the naive rightward only method previously described doubles the HiPPI traffic volume for no apparent reason; the data was already on the left machine and so there is no reason to return the data there.

What we did instead was have the node that is on the right edge of the source machine pass the message back to the left part of the same machine. One might argue that rather than give the nodes on the right edge this additional send, the nodes in the originating column could have done the job equally well. However, the combination of the pipelined nature of the code and the handshaking means that it makes more sense for nodes on the boundary to take care of the extra send (and avoid a performance penalty.) This alternative horizontal broadcast reduced the HiPPI traffic in half.

The next thing done to minimize HiPPI contention was combining multiple horizontal messages of different types into a single message. This enabled all horizontal messages to go through a single code layer. At each iteration of the code, only one rightward message was sent across each HiPPI node, paired with a single leftward handshake. All data, including data of different types (the double precision multipliers and the integer pivot indices), was combined into this single message. This optimization reduced N1/2 by nearly twenty five percent.

The code was also written so that the executable for the left machine could be used by itself to run a single machine problem.

2.2 MPLINPACK Modeling

We define the following variables:

N	=	MPLINPACK Problem Size (Matrix Order)
V	=	Number of vertical nodes (rows) in a logical 2D mapping
H	=	Number of horizontal nodes (columns)
K	=	Block size used in the algorithm
G	=	Node DGEMM FLOPS rate
T	=	Node DTRSM FLOPS rate (in operations / second)
U	=	K-Column local LU FLOPS rate
B	=	Communication Bandwidth (in bytes / second)
S	=	Communication Latency in terms of seconds
C	=	Average time to Copy long strided arrays of doubles in seconds

There are nine distinct places that MPLINPACK spends its time:

1. DGEMM Time:
   Assuming $K \ll N$,
   $$\frac{2N^3}{3VHG} \tag{1}$$
2. Horizontal Broadcast Time:
   Each row must send and receive its portion of the multipliers, which average $NK/2H$ doubles N/K times (once for each block.) For the HiPPI experiment, the B variable (for broadcast rate) should be portionally decreased to reflect the speed of moving horizontally through the HiPPI.
   $$2N/K * (\frac{8KN}{2VB} + S) \tag{2}$$
3. Vertical Broadcast Time:
   Each column must participate in a tree synchronization (log(V) messages) of a block averaging $NK/2Q$ doubles N/K times.
   $$\log(V) N/K * (\frac{8KN}{2HB} + S) \tag{3}$$
4. DTRSM Time:
   At any one time, DTRSM is done by only one row of nodes. Since all nodes in that column of nodes must wait, in our model we assume the

work is done redundantly. The amount of work is $K^2N/2H$ on average, done N/K times.

$$\frac{KN^2}{2HR} \tag{4}$$

5. K-Column local LU Work Time:
This is done by a single column of nodes at a time. On average there is $K^2N/2H$ FLOPS and N/KV local LUs performed by each column.

$$\frac{N^2K}{2HVU} \tag{5}$$

6. K-Column local LU Communication Time:
To find local pivot requirements, there is a tree fan in followed by a fan out, followed by the actual pivot just within the K column.

$$2\log(V) * (N/H)(8/B + S) + \log(V) * (N/H)(8K/B + S) \tag{6}$$

7. Pivoting Time:
The worst case is when when every pivot is off the node that owns the current block. Then one sends and receives all the rows $(2N)$ from the current column to the end of the matrix. This size averages $N/2V$. We need to also consider the cost of copying the $(2N)$ rows which average $N/2V$ size when all the elements are spread by at least (N/H) doubles.

$$2N * (\frac{8N}{2VB} + S + \frac{NC}{2V}) \tag{7}$$

8. Load Imbalance Delays:
Some nodes have more columns/rows than others in the DGEMM half the time. There are N/K DGEMMs total. If one node has K extra columns, there are $2K^2(N/2H)$ extra FLOPS (on average) associated with doing the DGEMM. If one node has K extra rows, this translates to an extra $2K^2(N/2V)$ FLOPS on average. We multiply this by N/K for the total number of times, and divide by two since it only occurs half the time.

$$\frac{N^2K}{2VG} + \frac{N^2K}{2HG} \tag{8}$$

9. Horizontal Waiting Delays:
The formula is the same as the horizontal time's formula. However, for the HiPPI experiment, the technique used to do local machine broadcasts indicates one should use the full memory bandwidth.

Plugging in $NR = 128600$, $V = 141$, $H = 16$, $K = 16$, $G = 134000000$, $T = 71000000$, $U = 11000000$, $B = 40-160000000$ (the smaller number only in Equation 2,) S as 30 microseconds, and $C = 3.4E-7$ yields a predicted performance of 277.832 GFLOPS [4]. This is close to our actual observed rate of 281.049.

3 Results

On 2256 MP nodes, across two MP Paragon supercomputers, we achieved 281.1 GFLOPS on MPLINPACK , which is 124.6 MFLOPS/node. This was done on a matrix of size 128600, with an N1/2 of 25700 [2]. The ZLU code achieved 328 GFLOPS on a complex matrix of size 90241.

A hypothetical single Paragon system with 2256 MP nodes would be likely to achieve approximately 126 MFLOPS/node on a problem of this size, so the result of 124.6 MFLOPS/node shows that the performance hit from distributing the problem was on the order of 1 percent.

4 Impact

The demonstration shows that distributed massively parallel computing is possible and feasible, and scalability can be achieved across multiple systems. Discussions are underway to connect Paragon systems at remote locations over hundreds or thousands of miles using the same software technology, to investigate the scalability characteristics.

Acknowledgements

Many people worked together to make this effort possible. The Sandia National Labs. scientists were Rolf Riesen, Gabi Istrail, Lance Shuler, Chu Jong, and the contractor LeeAnn Fisk of the University of New Mexico. The Intel scientists were Jerry Bolen, Bill Dazey, Arlin Davis, Satya Gupta, Greg Henry, David Robboy, Guy Schiffler, Mack Stallcup, Amir Taraghi, Stephen Wheat (formerly at SNL) and Bruce White. Indispensable support and assistance were provided by Natalie Bates, Bruce Blankinship, Walt Harrison, Dale Busacker, and others.

The record breaking demo used MPLINPACK. The MPLINPACK source base used at Intel was originally

[4] This ignores the final triangular solve, but that takes less than one tenth of one percent, even on a distributed machine such as was used here.

developed by Robert van de Geijn of the University of Texas at Austin. There was a joint Intel and Sandia MP LINPACK effort which proceeded the record breaking demo, and this was used to achieve the world record MPLINPACK performance in May 1994. Sandians involved in this, not mentioned above, include David Womble and David Greenberg. Intel scientists, not mentioned above, include Bob Norin. In addition, the main DGEMM [5] kernel for MPLINPACK was written by contractor Brent Leback, and parallelized by Greg Henry of Intel.

References

[1] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorenson, D., LAPACK Users' Guide, SIAM Publications, Philadelphia, PA, 1992

[2] Dongarra, J. J., *Performance of various computers using standard linear equations software in a Fortran environment.* Computer Science Technical Report CS-89-85, University of Tennessee, 1989

[3] Dongarra, J.J., Bunch, J., Moler, C., Stewart, G., *LINPACK Users' Guide*, SIAM Publications, Philadelphia, PA, 1979

[4] Dongarra, J. J., Du Croz, J., Hammarling, and Hanson, R., *An extended set of Fortran Basic Linear Algebra Subprograms.*, ACM Trans. Math. Software, 14:1-17, 1988

[5] Dongarra, J. J., Du Croz, J., Hammarling, and Duff, I. S., 1988, *A set of Level 3 basic linear algebra subprograms*, Report AERE R 13297. Oxford: Computer Science and Systems Division, Harwell Laboratory.

[6] Golub, G., Van Loan, C., Matrix Computations, 2nd Ed., 1989, *The John Hopkins University Press.*

[7] Hendrickson, B.A., Womble, D.E., *The torus-wrap mapping for dense matrix calculations on massively parallel computers.*, SIAM J. Sci. Stat. Comput., 1994

[8] Henry, G., *BLAS Based on Block Data Structures*, Theory Center Technical Report, CTC92TR89, 1/92.

[9] Henry, G., *Increasing Data Reuse in Eigenvalue Related Problems*, Ph.D. Thesis, Cornell University, January 1994

[10] Henry, G., *The COP Interface and MP Linpack: A Simple Users Guide*, Intel SSD Technical Report in Progress, August 1994

[11] Intel Supercomputer Systems Division, *Intel Paragon XP/S Technical Summary*, Document SSD9401R13N, May 1994

[12] Kågström, B., Van Loan, C.F., *GEMM-Based Level-3 BLAS*, Theory Center Technical Report, CTC91TR47, 1/91.

[13] Pandit, M., *VCF: A Connection Based Message-Passing Facility*, 1994 Annual Users' Conference Proceedings, Intel Supercomputer Users Group, San Diego, CA, 6/94

[14] Strassen, V., *Gaussian Elimination is not Optimal*, Numer. Math. Vol. 13, 1969, pp. 354–356

[15] van de Geijn, R.A., *Massively Parallel LINPACK Benchmark on the Intel Touchstone DELTA and iPSC(R)/860 Systems.*, 1991 Annual Users' Conference Proceedings. Intel Supercomputer Users' Group, Dallas, TX, 10/91

[16] Winograd, S., *A new algorithm for inner product*, IEEE Trans. Comp., Vol. C-37, 1968, pp. 693–694

[17] Womble, D.E., Greenberg, D.S., Wheat, S.R., Riesen, R., *LU Factorization and the LINPACK benchmark on the Intel Paragon*, Sandia Technical Report, 1994.